Series: Monografias em Ciência da Computação,
No.24/89

ON THE INHERENT NON-MONOTONICITY OF SOFTWARE DEVELOPMENT: A FORMAL JUSTIFICATION

Armando M. Haeberer
Paulo A. S. Veloso

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453
RIO DE JANEIRO - BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Series : Monografias em Ciência da Computação No. 24 / 89

Series Editor : Paulo A. S. Veloso October 1989

ON THE INHERENT NON - MONOTONICITY OF SOFTWARE DEVELOPMENT : A FORMAL JUSTIFICATION

Armando M. Haeberer *

Paulo A. S. Veloso

+ Research partly sponsored by the ETHOS project of the Argentine-Brazilian Program of Research and Advanced Studies in Computer Science and by FINEP.

* On leave from
ESLAI : Escuela Superior Latinoamericana de Informática
PO Box 3193 , 1000 Buenos Aires ; ARGENTINA

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC RIO, Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
BRASIL

Tel.:(021)529-9386 TELEX:31078 FAX:(021)274-4546
BITNET: userrtlc@lncc.bitnet

ABSTRACT

The software development process goes from a verbalization of the requirements of a real problem to a final program, perhaps via a formal specification. Its goal is that the virtual machine described by the program behaves as an engineering model of the real problem (application concept). Here, these objects, as well as some interconnections among them, are analyzed from a theoretical standpoint by means of Carnap's Two-Level Theory of the Language of Science and the Algebraic Theory of Problems.
The relation "being-an-engineering-model" is argued to be synthetic and a disposition, which has interesting consequences for factorizations of the process. On the one hand, specification validation and program testing are inevitable, but the latter must be preceded by some correctness verification. On the other hand, the software process exhibits an inherent non-monotonocity, both globally and locally.
This formal framework enables us to clarify the status of the objects involved and their interconnections. We are able to state, and establish, in a precise way some facts that are generally believed on intuitive grounds only.

Keywords : Software development process, software engineering, formal specifications, correctness, specification validation, program testing, non-monotonocity, epistemology, theory of problems.

RESUMO

O processo de desenvolvimento de programas vai da verbalização dos requisitos de um problema real a um programa final, talvez via uma especificação formal. Seu objetivo è que a màquina virtual descrita pelo programa se comporte como um modelo "engenheril" do problema real (conceito de aplicação). Aqui, estes objetos, bem como algumas relações entre eles, são analisados de um ponto de vista teòrico por meio da Teoria dos Dois Nìveis da Linguagem da Ciência de Carnap e da Teoria Algèbrica de Problemas.
Argumenta-se que a relação de ser-um-modelo-"engenheril" è sintètica e uma disposição, o que tem conseqüências interessantes para fatorações do processo. Por um lado, validação de especificações e teste de programas são inevitàveis, mas este deve ser precedido por alguma verificação de corretude. Por outro lado, o processo de programção exibe uma não monotonia inerente, tanto a nìvel global quanto local.
Este arcabouço formal permite clarificar a natureza dos objetos envolvidos e suas relações. Conseguimos enunciar, e demonstrar, de maneira precisa alguns fatos que são geralmente aceitos apenas por razões intuitivas.

Palavras chave : Processo de desenvolvimento de programas, engenharia de software, especificações formais, corretude, validação de especificações, teste de programas, não monotonia, epistemologia, teoria de problemas.

RESUMEN

El proceso de desarrollo de software va desde la verbalización de los requerimientos de un problema real hasta un programa final, quizás via una especificación formal. Su objetivo es que la máquina virtual descripta por el programa se comporte como un modelo ingenieril del problema real (concepto de aplicación). Aquí se analizan estos objetos, así como algunas relaciones entre ellos, desde un punto de vista teòrico por medio de la Teoría de los Dos Niveles del Lenguaje de la Ciencia de Carnap y de la Teoría Algebraica de Problemas.
Se argumenta que la relaciòn "ser-un-modelo-ingenieril" es sintètica y una disposiciòn, lo cual tiene consecuencias interesantes para factorizaciones del proceso. Por una parte, la validaciòn de especificaciones y el testeo de programas son inevitables, pero èste debe ser precedido por alguna verificaciòn de corrección. Por otra parte, el proceso de programaciòn exhibe una no monotonìa inherente, tanto a nivel global como local.
Este marco formal pemite aclarar la naturaleza de los objetos involucrados y sus relaciones. Logramos enunciar, y demostrar, de manera precisa algunos hechos que son generalmente aceptados sòlo por razones intuitivas.

Palabras clave : Proceso de desarrollo de software, inginería de software, especificaciones formales, corrección, validaciòn de especificaciones, testeo de programas, no monotonìa, epistemologìa, teoría de problemas.

I. INTRODUCTION.

In this paper we will analyze from a formal standpoint how requirements, informal and formal specifications and programs relate among themselves, focusing on the roles of validation and verification and on the inherent non-monotonicity of software development.

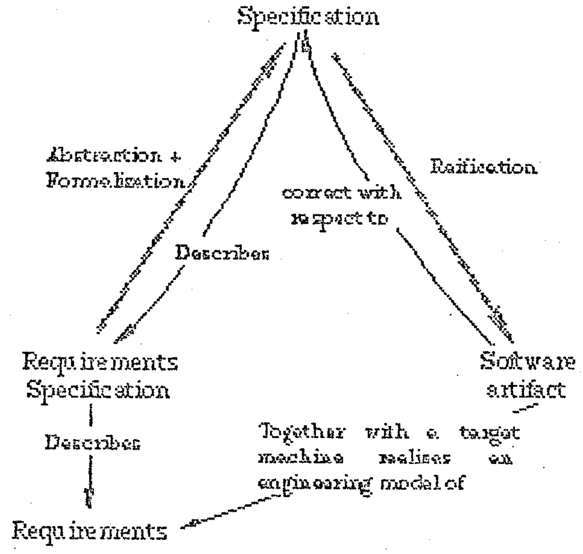

Figure I.1.

Software development, at a high level of abstraction, may be regarded as involving four main objects. They are the requirements, reflecting the *real problem* extension; the requirement specification, as the informal description of the requirements; the formal specification[1], as its formal description; and the program giving rise to a virtual machine, as the final product of the process.

A formal analysis requires a formal framework. This framework should have two main goals. First, its formalism should allow the treatment of the software process and its components at any level of abstraction. Second, its formal structure should shed light on concepts like validation, verification, informal description, formal specification, abstraction, etc. We will use as formalism the Algebraic Theory of Problems [Vel84; HVE83], and as formal structure Rudolf Carnap's Two-Level Theory of the Language of Science [Car36, 56; Ste70].

Some terms related to software processes, software engineering, etc., are not very well defined. In order to achieve precision, we will resort to a terminology like that of [Leh84], instead of the "classical" software engineering. So, we will use application concept for requirements, verbalization for requirement specification, virtual machine for implemented program, etc.

Within this formal framework we will, then, analyze relations such as "describes", "realizes an engineering model", "is correct", etc., as well as the validation and proof obligations involved in factorizing the diagram in figure I.1.

[1] A common misconception is that of considering formal specification only as formulas in some mathematical language. But, an abstract program is a formal, though somewhat odd, specification. Furthermore, a rapid prototype is a formal specification, probably incomplete or wrong in its early stages, but ovicusly formal.

II. THE SOFTWARE PROCESS.

The goal of the software process is the construction of a software artifact. This construction starts from a description of some *real problem*. The software artifact, together with a given target machine, is to behave as an engineering model of the *real problem*. Although this statement is quite informal and vague - which we will try to fix in the sequel- it suffices to show the point we are trying to emphasize here, namely the enormous difficulty in achieving the goal. This difficulty stems from the fact that the description of the *real problem* is usually vague, informal, ambiguous, and lacking in details, whereas the software artifact is a complex syntactic object belonging to a formal language.

Thus, the software process ranges over a wide spectrum of activities, such as formalization, abstraction, interpretation, construction of solutions, development of algorithms, validation, verification, etc. In addition, the lack of precision in the initial description of the *real problem* prevents this process from being linear. Backtracking occurs frequently due to, for instance, the use of the formality and precision inherent to specification and programming languages as heuristic tools for the understanding and clarification of the initial description.

II. 1. The application-concept and its verbalization. Two terms related to the *real problem*, application concept and verbalization, should be clarify. By *application concept* we mean the extensional knowledge about the *real problem* that serves as reference point of the software process. By a *verbalization* of the application concept $\mathbb{A}$ we mean a meta-linguistic description $\mathcal{V}_{\mathbb{A}}$ of this application concept.

Application concept is the information about the *real problem*, as detailed and clear as one understands it at a given moment. Thus, an application concept may be vague, not detailed, etc. But it may also be as precise and detailed as one wishes; such would be the case if one knew exactly the extension of the *real problem*. So, an application concept may be initially wrong and may be corrected by backtracking in view of the heuristic power of the development process of a software artifact; it becomes completely determined at process completion.

On the other hand, a verbalization of an application concept amounts to its description in the meta-language. It represents the starting *communicable* comprehension of an application concept. So, a verbalization will be incomplete if the application concept is incomplete, but no matter how detailed and precise is the latter, ambiguities and fuzziness of the meta-language carry over to verbalizations expressed in it.

Since the application concept is an extensional object, it cannot be an informal one. The application concept is an observable object, which can be ill-determined because of lack of knowledge about its extension. Its verbalization, from which the software process starts, is the informal and ambiguous object that is often mistaken for the application concept itself.

We will say that a *input δ belongs to the domain of* $\mathbb{A}$, denoted by $\mathfrak{D}\delta\mathbb{A}$, iff δ designates an acceptable input for $\mathbb{A}$, and that the ordered pair $\langle\delta,\rho\rangle$ is an *instance of* $\mathbb{A}$, denoted by $\mathcal{I}\delta\rho\mathbb{A}$, iff δ belongs to the domain of $\mathbb{A}$ and ρ designates an acceptable output for $\mathbb{A}$ corresponding to input δ. We will accept as input and output any pair of *observable events* related by the application, in the sense of belonging to the extension of the *real problem*.

II.2. The synthetic [1]content of the software development process. A software artifact is a program or a set of programs written in some programming language to be interpreted by a target machine. So a software artifact is a formal syntactic object that, upon interpretation by a target machine, realizes a device that accepts inputs δ and produces outputs ρ.

By *target machine* we mean a device, made out of hardware and software, that is capable of interpreting a program p. We shall generally use H to refer to a target machine. On the other hand, by *virtual machine* we mean the device m_{pH} constructed by interpreting a software artifact p on a target machine H. By the *result of virtual machine* m_{pH} *for data* δ *at instant* t, denoted by $m_{pH}t(\delta)$, we mean the output [2] ρ produced by m_{pH} at instant t after being fed data δ, provided that it halts.

We will now try to state in precise terms the meaning of the relation of being-an-engineering-model. This relation, denoted by $m_{pH} \angle A$, connects a virtual machine m_{pH} to an application concept A. Clearly, the observation of $m_{pH} \angle A$ presupposes a systematic activity. This activity can be described as follows. First, an input δ from the domain of A is selected and introduced into machine m_{pH}.

Then, if m_{pH} does not halt, it is not the case that $m_{pH} \angle A$. If m_{pH} does halts but $\neg(\mathcal{I}\delta m_{pH}t(\delta)A)$ then $\neg(m_{pH} \angle A)$; if m_{pH} halts and $\mathcal{I}\delta m_{pH}t(\delta)A$ then we may assume $m_{pH} \angle A$.

Let us establish some abbreviations, namely $\mathcal{A}m_{pH}\delta$ for machine m_{pH} is applied to input δ, $\mathcal{H}m_{pH}\delta t$ for machine m_{pH} on input δ halts at instant t. We can now give a first precise definition of being -an-engineering-model.

$$m_{pH} \angle A \leftrightarrow (\forall\delta)[\mathcal{E}\delta A \wedge \mathcal{A}m_{pH}\delta \rightarrow (\exists t)(\mathcal{H}m_{pH}\delta t \wedge \mathcal{I}\delta m_{pH}t(\delta)A)] \qquad (0)$$

Why have we stated "we may assume $m_{pH} \angle A$" instead of directly "$m_{pH} \angle A$"? The answer resides in the asymmetry of factual hypotheses. To understand it we must realize the factual character of (0). It stems from the presence of a universal quantification over the set of inputs δ belonging to the application domain. Except in some very special cases, this set is not exhaustible, and any attempt to define it by comprehension will be marred by the informality and ambiguity of the meta-language. So, one can never actually fulfill the condition "for all input δ" in the definiens of (0). One generally induces this condition from a certain sample. Hence, (0) is in fact a synthetic statement in the sense of empirical science.

The hypothetico-deductive method [Hem65] then suggests an experiment[3] for it, which we can described naively as follows.

We take $m_{pH} \angle A$ as main hypothesis and add the auxiliary hypothesis $\bigwedge_{i=1}^{n} (\mathcal{E}\delta_i A \wedge \mathcal{A}m_{pH}\delta_i)$. From the latter we derive the observational consequence $\bigwedge_{i=1}^{n} [(\exists t)(\mathcal{H}m_{pH}\delta_i t) \wedge \mathcal{I}\delta_i m_{pH}t(\delta_i)A)]$. If the experiment re-

1 We shall be dealing with analytically determinate and synthetic statements. We gloss over some polemical issues and assume that every meaningful scientific statement can be classify as either analytically determinate or synthetic [Ste70].

The class of analytically determinate statements includes those statements whose truth can be determined by a mere analysis of meaning. In this class we find the purely logico-formal truths and the logical falsities, i.e., statements whose truth or falsity is completely determined by the meaning of the logical symbols (connectives, quantifiers, etc.). To these we add the analytical truths (consequences of statements where the meaning relations among descriptive expressions are fixed) and their negations (the analytical falsities). The synthetic statements are those that are not analytically determinate; their truth can be analyzed only experimentally. Thus the synthetic statements can be identified with the empirically determinate ones.

2 We regard "output", "input", "halts", etc., as primitive terms whose meaning is assumed to be understood.

3 The existential quantifier in the observational consequence will be discussed later.

jects the observational consequence, then we have rejected the conjunction of the hypotheses. As we assume the auxiliary hypothesis true by construction, we refute $\mathfrak{m}_{pH} \angle \mathbf{A}$. If, on the other hand, the experiment does not falsify the observational consequence, we are not entitled to deduce $\mathfrak{m}_{pH} \angle \mathbf{A}$, for we would be resorting to a well-known fallacy.

In this case, we can accept $\bigwedge_{i=1}^{n} [(\mathfrak{E}\delta_i\mathbf{A} \wedge \mathcal{A}\mathfrak{m}_{pH}\delta_i) \wedge (\exists t)(\mathcal{H}\mathfrak{m}_{pH}\delta_i t) \wedge \mathcal{I}\delta_i\, \mathfrak{m}_{pH}\, t(\delta_i)\, \mathbf{A}))]$, only as a good inductive support for $\mathfrak{m}_{pH} \angle \mathbf{A}$.

III. THE DISPOSITIONAL CHARACTER OF THE SOFTWARE PROCESS GOAL.

Formula (0) is an example of an operational definition. This name arises from the fact that the definiens of such definitions involves an "operation", in our case the application of machine $\mathfrak{m}_{pH}$ to each data δ. This kind of definition appears to capture our intuition. The properties we are dealing with involve systematic observation of reactions to operations. Thus, it seems reasonable to use such operations and reactions in a definiens. Properties (like "soluble", "breakable", "magnetic") whose observation involves a systematic activity are called *dispositions*. Operational definitions were proposed to introduce such dispositions into the language of science.

Unfortunately, as Carnap showed in a now classical argument [Car36], operational definitions are too wide and so fail to accomplish their goal. Let us point out this problem in (0). Our intuition was based on the fact that test conditions $\mathfrak{E}\delta\mathbf{A}$ and $\mathcal{A}\mathfrak{m}_{pH}\delta$ were true. But what if these conditions did not hold? Then, the antecedent of the definiens of (0) would be false and, hence, the implication true. Consider a machine $\mathfrak{m}_{pH}$ that has never been tested with respect to application concept $\mathbf{A}$. We then have $(\forall\delta)(\mathfrak{E}\delta\mathbf{A} \rightarrow \neg\mathcal{A}\mathfrak{m}_{pH}\delta)$, hence $(\forall\delta)(\mathfrak{E}\delta\mathbf{A} \wedge \mathcal{A}\mathfrak{m}_{pH}\delta \rightarrow (\exists t)(\mathcal{H}\mathfrak{m}_{pH}\delta t) \wedge \mathcal{I}\delta\mathfrak{m}_{pH} t(\delta)\, \mathbf{A}))$, i.e. $\mathfrak{m}_{pH} \angle \mathbf{A}$. In other words, from (0) we can conclude that a virtual machine that has never been tested with respect to an application concept $\mathbf{A}$ turns out to be an engineering model of $\mathbf{A}$! This is a troublesome feature of operational definitions.

III.1. The introduction of dispositions into the language of science. Directly observable properties, such as "red", "liquid", etc. appear in the language of science as primitive predicates. For any systematic construction of science, one should have few primitive predicates. On the other hand many properties are dispositions[1] [Ste70]. Thus, one cannot take all dispositional predicates as primitive, if one wishes a simple system. As mentioned, the use of operational definitions to introduce dispositional predicates into the language of science presents problems.

One of the approaches to the introduction of dispositions was Carnap's proposal of replacing operational definitions by reductive sentences.. Consider, for instance, the disposition introduced in (0). It would be replaced by the pair:

$$(\forall\delta)[\mathfrak{E}\delta\mathbf{A} \wedge \mathcal{A}\mathfrak{m}_{pH}\delta \rightarrow (\mathfrak{m}_{pH} \angle_{\delta} \mathbf{A} \leftrightarrow ((\exists t)(\mathcal{H}\mathfrak{m}_{pH}\delta t \wedge \mathcal{I}\delta\mathfrak{m}_{pH}\, t(\delta)\mathbf{A}))] \qquad (1)$$

$$\mathfrak{m}_{pH} \angle \mathbf{A} \leftrightarrow (\forall\delta)(\mathfrak{E}\delta\mathbf{A} \rightarrow \mathfrak{m}_{pH} \angle_{\delta} \mathbf{A}) \qquad (2)$$

The value of this modification can be assessed from distinct viewpoints. From the purely logical viewpoint, (1) amounts to a conditional definition: the definiendum is related to the definiens under the condition $\mathfrak{E}\delta\mathbf{A} \wedge \mathcal{A}\mathfrak{m}_{pH}\delta$. From the viewpoint of philosophy of science, there is a great difference between (1 & 2) and an operational definition. Under the light of the principle of partial interpretation of the the-

[1] It seems that all the proprieties of virtual machines are dispositions; consider complexity, efficiency, etc.

oretical terms (which will be discussed later on), reductive sentences are already cases of a partial axiomatic characterization of a dispositional term.

The inadequacy of operational definitions, in the sense of being too wide, disappears upon their replacement by reductive sentences. If condition $\mathfrak{E}\delta\mathfrak{A} \wedge \mathfrak{A}\mathfrak{m}_{pH}\delta$ does not hold, the previous puzzling conclusion is now replaced by an indetermination about the presence of the disposition. On the other hand, a definition must satisfy the so called "eliminability principle": the definiendum can be replaced by the definiens in every context. Reductive sentences do not satisfy this principle, but this is one of the requirements abandoned by Carnap in introducing two levels in the language of science [Car56].

Two new insurmountable difficulties arose at this point. The first difficulty is the matchlessness of the reductive sentence method with actual behavior of a working scientist facing a negative result of an experiment. A researcher does not reject a disposition just because of a single negative experimental outcome. For instance, he may suspect that certain disturbing conditions render the experiment unreliable. From this standpoint, an experiment permits an exception clause, which reductive sentences do not.

III.2. The decidability of observational terms. Let us discuss now the second difficulty. To clarify the context of the following discussion, it would be helpful to realize that before Carnap's Two-Level Theory, the whole language of science was considered to be a global empiricist language, denoted $\mathcal{L}_E$. This language included all the terms of science, both observable and theoretical ones.

One of the two languages proposed by Carnap is the observational language $\mathcal{L}_O$, which we will discuss in the sequel. Let us now turn our attention to the confirmability and refutability in principle of the terms of $\mathcal{L}_O$. We can classify terms definable by means of an observational vocabulary according to their definitions. Observationally decidable terms are introduced by definitions providing explicit criteria for confirming or refuting a property; the remaining terms are observationally undecidable.

From the standpoint of observational decidability, the class of terms of $\mathcal{L}_O$ can be divided as follows. On the one hand, the *observationally decidable terms*, introduced by means of primitive predicates of $\mathcal{L}_O$ (expressing directly observable properties) and predicates whose definiens must not contain quantifiers. On the other hand, *observationally undecidable terms*, introduced by means of predicates whose definiens may contain quantifiers with potentially infinite domain. This class can be subdivided into: predicates whose definiens contain only universal quantifiers (non-confirmable, but refutable, in principle), predicates whose definiens contain only existential quantifiers (non-refutable, but confirmable, in principle), and predicates whose definiens contain both kinds of quantifiers (neither confirmable nor refutable in principle).

Thus, the predicate $(\exists t)(\mathfrak{H}\mathfrak{m}_{pH}\delta t \wedge I\delta\mathfrak{m}_{pH}t(\delta)\mathfrak{A})$ is not $\mathcal{L}_O$-decidable, which can be easily seen as follows. If machine $\mathfrak{m}_{pH}$ halts at time t yielding a result $\mathfrak{m}_{pH}t(\delta)$, one is able to examine whether $I\delta\mathfrak{m}_{pH}t(\delta)\mathfrak{A})$ holds. If, on the other hand, machine $\mathfrak{m}_{pH}$ has not yet halted at time t, one is unable to know whether or not it would do so at time $t+1$; and this holds whenever $\mathfrak{m}_{pH}$ has not halted.

IV. THE TWO LEVEL THEORY OF THE LANGUAGE OF SCIENCE.

The difficulties discussed above led Carnap [Ste 70] to abandon both the idea of a global empiricist language $\mathcal{L}_E$ and the need to introduce dispositional predicates into the language only by means of reductive sentences. Then, Carnap proposed the Two-Level Theory of the Language of Science [Ste70].

IV. 1. Observational and Theoretical Levels in Science. Carnap proposed to split the language of science into two languages. One part is the basic empiricist language, understandable by itself, which he called observational language $\mathcal{L}_O$. The other part, called the theoretical language $\mathcal{L}_T$, is the language for formulating a theory. The latter language is not understandable by itself and does not have a complete empirical interpretation. A partial empirical interpretation is obtained by means of a set C of correspondence rules connecting some extralogical expressions of $\mathcal{L}_T$ to expressions of the observational language. Some dispositions may be regarded as closer to theoretical concepts than to observational ones; as such, they should be introduced into the theoretical language. Furthermore, by means of the connection between $\mathcal{L}_O$ and $\mathcal{L}_T$ one can take into account the exception clause needed for a "reasonable" treatment of negative experimental outcomes.

Let $\mathbb{M}$ be a (dispositional) concept (of $\mathcal{L}_T$) [Car56]. Consider the following hypothetico-deductive scheme. Let H_M be a hypothesis about the presence of $\mathbb{M}$. We add two sets of auxiliary hypotheses, namely: H_X a set of theoretical hypotheses and H_1 a set of some descriptive observational statements. Then, by using the underlying theory T and the set C of correspondence rules, we derive an observational consequence Oc.

We will have the meta-theoretical statement $H_M \wedge H_X \wedge H_1 \wedge T \wedge C \vdash Oc$. Assume now that the expected observational consequence Oc does not occur. Then, $\neg Oc \wedge H_X \wedge H_1 \wedge T \wedge C \vdash \neg H_M$. Nevertheless, in contrast to the case of reductive sentences, even though $\neg Oc$, one may keep theory T and correspondence rules C, without having to accept $\neg H_M$. One may instead assume that some theoretical hypotheses of H_X or some descriptive observational statements of H_1 are false[1].

Language $\mathcal{L}_O$ is an extensional, completely interpreted language, whose alphabet V_O is the observational vocabulary. Then all the predicates of V_O designate observable properties of events or things. Carnap uses $\mathcal{L}_O$ as what he calls the restricted observational language, which has only directly observable properties. Here, we use an extended observational language $\mathcal{L}_O^*$, which allows the introduction of $\mathcal{L}_O$-definable dispositional predicates (see [Car56] section IX and [Ste70]).

Some requirements are imposed on an observational language $\mathcal{L}_O^*$. $\mathcal{L}_O^*$ must be a non-modal language with at least one finite model; in addition, its primitive descriptive terms must be observable, the values of each one of its variables must be designated by an expression of $\mathcal{L}_O^*$ and every concept introduced must be reducible to the primitive descriptive terms.

The primitive symbols of $\mathcal{L}_T$ are divided into logical and descriptive (or extralogical) symbols. Hence the theoretical vocabulary V_T will be the class of all theoretical symbols In general, it is not possible to give explicit definitions for such symbols on the basis of $\mathcal{L}_O^*$. A theoretical language $\mathcal{L}_T$ is designed to have all the freedom needed. A Theory T in $\mathcal{L}_T$ consists of a finite number of postulates formulated in $\mathcal{L}_T$; as such, it is an uninterpreted calculus. Objects of $\mathcal{L}_O^*$ have no formal denotations, their meaning is given directly by observation, in view of requirements of observabilty and reducibility imposed on $\mathcal{L}_O^*$.

[1] We identify a finite set of formulas with their conjunction, when conveninient.

Correspondence rules connect some terms of V_T to those of V_O, in order to provide the partial interpretation of $\mathcal{L}_T$ in terms of $\mathcal{L}_O^*$. Therefore, we are free to choose the logical structure of $\mathcal{L}_T$ to suit our needs. Now, theory T becomes an interpreted calculus, for it comes together with a set C of correspondence rules. A scientist will use the partially interpreted theory $T \wedge C$ to guide his expectations by deriving predictions about the behavior of observable objects. The rules in C allow the derivation of certain sentences of $\mathcal{L}_O^*$ from certain sentences of $\mathcal{L}_T$ and vice versa; we always use the path through $\mathcal{L}_T$ to obtain a sentence of $\mathcal{L}_O^*$ from certain sentences of $\mathcal{L}_O^*$. We will use the abbreviation C(...) to express the result of applying the rules of C to an object ... belonging either to V_T or to V_O.

IV.2. Observational and Theoretical Objects in the Software Process. The observational level for the software process comprises application concepts $\mathcal{A}$ and virtual machines m_{pH}. Their inputs δ and outputs ρ belong to a universe $\mathcal{W}$. In addition, we also have in this level a denumerable ordered set called *time*. So, statements such as $\exists\delta\mathcal{A}$ and $\mathcal{A}m_{pH}\delta$ belong to $\mathcal{L}_O^*$. Statements like $m_{pH} \angle \mathcal{A}$ and $\mathcal{H}m_{pH}\delta t$ are also formulated in $\mathcal{L}_O^*$ but must be dealt with in $\mathcal{L}_T$.

The theoretical level is split into two layers, the syntactical and the semantical one. The syntactical layer has programs and specifications; the semantical layer comprises their denotations, problems, as will be explained shortly. Then, the Algebraic Theory of Problems, Set Theory, the Fixpoint Theory of programs, etc., will all be part of the theoretical level.

These rules relates observational inputs δ and outputs ρ to their theoretical counterparts d and r. Thus, they correlate denotations of specifications and observational objects (i.e. application concepts).

The target machine H plays a special role in that it provides a set H of correspondence rules that connect a program p to its virtual machine m_{pH}, as such they correspond to an interpreter. The virtual machine gives the extensional behavior of its program. Thus, the latter is a description by comprehension of the former. Likewise a specification describes by comprehension an application concept, which within the observational level can be accessed only by extension (in a pointwise manner).

At this point one should recall that, as stated earlier, an application concept $\mathcal{A}$ stands here for the extension of a *real problem* and not necessarily for its description by comprehension, i. e. by means of a property. In some rare cases the application concept has a rather simple extension or can be described by comprehension in the observational language. Otherwise, the starting point of the software development process is an application verbalization $\mathcal{V}_{\mathcal{A}}$, which is a sentence of the meta-language. So, if, as it often happens, the meta-language is ambiguous, the notation $\mathcal{V}_{\mathcal{A}}$ is not quite correct, for it describes not exactly one but a class of application concepts $K_{\mathcal{V}}$.

V. A PROBLEM-THEORETIC APPROACH TO SOFTWARE DEVELOPMENT.

The concept of problem and a General Theory of Problems were developed from the ideas of G. Polya [Pol57] by P. A. S. Veloso [Vel84]. The goal of this development is a formal tool for reasoning about problem solving and modeling various strategies, techniques, methods, etc. On the basis of that theory, an Algebraic Theory of Problems [HBV87] was developed, aiming at a tool for the formal treatment of the software development process at various levels, ranging from the purely epistemological one of process explication [H+V89a, b], through those of prescribing different process obligations [V+H89] and modeling programming methods [Zar88], to a calculus for program derivation [EVHV89; V+E89; HVE89].

In the sequel we will outline a version of the Algebraic Theory of Problems; for more details, see [HVE89].

V.1 The Algebraic Theory of Problems. A *problem* over a universe $\mathcal{U}$ is a 3-tuple $\mathbb{P} = \langle D_{\mathbb{P}}, R_{\mathbb{P}}, \mathcal{P} \rangle$ where $D_{\mathbb{P}}$ and $D_{\mathbb{P}}$ are subsets of $\mathcal{U}$ and $\mathcal{P} \subseteq D \times R$. This mathematical structure attempts to capture the essence of the three questions suggested by Polya in approaching a problem: *what are the data? what are the results?* and *what is the problem condition?* Hence, in the 3-tuple $\mathbb{P} = \langle D_{\mathbb{P}}, R_{\mathbb{P}}, \mathcal{P} \rangle$, $D_{\mathbb{P}}$ stands for the *data domain*, $R_{\mathbb{P}}$ for the *result domain* and $\mathcal{P}$ for the problem *condition*[1] (also denoted by $q_{\mathbb{P}}$). We call a problem *deterministic* if its condition is a function. We say that problem $\mathbb{P} = \langle D_{\mathbb{P}}, R_{\mathbb{P}}, \mathcal{P} \rangle$ is *viable*, denoted by $\mathcal{V}i\mathcal{b}\mathbb{P}$, iff $(\forall d)\,(d \in D \rightarrow (\exists r)\,(r \in R \wedge \mathcal{P}(d, r)))$. Notice that $\mathcal{V}i\mathcal{b}\mathbb{P}$ is equivalent to the equality of $\mathcal{D}om\,\mathcal{P}$ and $D_{\mathbb{P}}$.

The 3-tuple representation captures the idea of choice associated with obtaining an acceptable result for each given data in a stated, but still unsolved, problem. Then, a solution should be an object that eliminates this choice and, therefore, solving a problem should mean constructing such an object.

Some *algorithmic operations* on problems were defined.

Given problems $\mathbb{P}$ and $\mathbb{Q}$, we define their *sum* $\mathbb{P} + \mathbb{Q} = \langle D_{\mathbb{P}} \cup D_{\mathbb{Q}}, R_{\mathbb{P}} \cup R_{\mathbb{Q}}, \mathcal{P} \cup \mathcal{Q} \rangle$, their *product* $\mathbb{P} * \mathbb{Q} = \langle D_{\mathbb{P}}, R_{\mathbb{Q}}, \mathcal{P}/\mathcal{Q} \rangle$ and their *direct product* $\mathbb{P} \times \mathbb{Q} = \langle D_{\mathbb{P}} \times D_{\mathbb{Q}}, R_{\mathbb{P}} \times R_{\mathbb{Q}}, \{ \langle\langle d, r\rangle, \langle d', r'\rangle\rangle : \langle d, r\rangle \in \mathcal{P} \wedge \langle d', r'\rangle \in \mathcal{Q} \} \rangle$. We will denote by $\mathbb{P}^{*n}$ the product of $\mathbb{P}$ by itself n times,.

By resorting to the generalized union and Cartesian product, we extend the binary operations sum and direct product to classes of problems, yielding *summation* and *generalized direct product*. We will use $\mathbb{P}^{\times n}$ to denote the direct product of n copies of $\mathbb{P}$. Along these lines, $\mathbb{P}^{**}$ and $\mathbb{P}^{\times *}$ denote the closure of $\mathbb{P}$ under product and direct product, respectively.

Some *non-algorithmic operations* on problems were also defined.

Given problems $\mathbb{P}$ and $\mathbb{Q}$, we define their *difference* $\mathbb{P} - \mathbb{Q} = \langle D_{\mathbb{P}} - D_{\mathbb{Q}}, R_{\mathbb{P}} - R_{\mathbb{Q}}, (\mathcal{P} - \mathcal{Q}) \cap (D_{\mathbb{P}} - D_{\mathbb{Q}} \times R_{\mathbb{P}} - R_{\mathbb{Q}}) \rangle$ (where $-$ stands for the difference of sets) and the *inverse* $\mathbb{P}^{-1} = \langle R_{\mathbb{P}}, D_{\mathbb{P}}, \mathcal{P}^{\smile} \rangle$, where $\mathcal{P}^{\smile}$ is the *converse* of $\mathcal{P}$.

Some important relations between problems were defined. Given problems $\mathbb{P}$ and $\mathbb{Q}$, we say that $\mathbb{P}$ is a *relaxation* of $\mathbb{Q}$ (denoted $\mathbb{P} \lrcorner\, \mathbb{Q}$) iff $D_{\mathbb{Q}} \subseteq D_{\mathbb{P}}$ and $\mathcal{P}|_{D_{\mathbb{Q}}} \subseteq \mathcal{Q}$, $\mathbb{P}$ is a *subproblem* of $\mathbb{Q}$ (denoted $\mathbb{P} \subseteq \mathbb{Q}$) iff there exist a problem $\mathbb{R}$ such that $\mathbb{P} + \mathbb{Q} = \mathbb{R}$, and $\mathbb{P}$ is a *complete subproblem* of $\mathbb{Q}$ (denoted $\mathbb{P} \subseteq_c \mathbb{Q}$) iff $\mathbb{P} \subseteq \mathbb{Q}$ and $D_{\mathbb{P}} = D_{\mathbb{Q}}$. The relations $\lrcorner$, $\subseteq$ and $\subseteq_c$ are transitive. Notice that $\subseteq_c$ and $=$ (the equality between problems) are special cases of relaxation. However they deserve attention because they have better monotonicity properties. The decision of restricting the correctness relation to one of them is part of the particular software construction strategy being used.

Let be given a set $\mathfrak{G}$ of special problems, which will be called *easy problems* and a set $\mathbb{G}$ of distinguished problems. By the *algebra of problems* over $\mathfrak{G}$ and $\mathbb{G}$ we mean the algebra $\mathcal{A} = \langle \mathbb{P}, \mathbb{G}, \mathfrak{G}, \{+, *, \times, -, \Sigma, {}^{-1}, {}^{\times n}, {}^{\times *}, {}^{**}\} \rangle$, where $\mathbb{P}$ is the set of all the problems over $\mathcal{U}$.

V.2. Specifications, problems, programs and solutions. A program should be interpretable by a *target machine*. As such, it should resort only to algorithmic operations. We cannot pretend that a target machine can provide a computation for an arbitrary problem. In fact, a general-purpose target machine in-

[1] We employ the usual notations for relations: $\mathcal{D}om\ \mathfrak{R}$ and $\mathcal{R}an\ \mathfrak{R}$ stand for the *domain* and *range* of the relation $\mathfrak{R}$. / stands for the relative product of relations. The converse of $\mathfrak{R}$ is the relation $\mathfrak{R}^{\smile}$ such that for all x and y, $x\mathfrak{R}^{\smile}y$ iff $y\mathfrak{R}x$ [Tar41]

terprets a finite, and usually not too large, set of constant symbols over problems ; their denotations form our set $\mathfrak{P}$.

Problems are on the semantical layer. We obtain the syntactical layer by introducing symbols for the constants and operations of the algebra $\mathcal{A}$. We will consider a language $\mathcal{L}$, called *global language*, with symbols[1] for the constants and operations of $\mathcal{A}$. Let us denote by $\mathbb{T}$ the set of terms of $\mathcal{L}$ and by $\mathcal{Æ}$ the corresponding algebra of terms generated from $\mathfrak{P} \cup \mathfrak{C}$ [GTW78]. As usual there is a unique homomorphism $\mu : \mathcal{Æ} \to \mathcal{A}$, assigning a problem as value to each term. Such a term $T \in \mathbb{T}$ is a specification and the semantic function μ assigns to T the problem it describes.

A constant symbol f, denoting a problem $\mu[f]$, is said to be *easy* with respect to target machine $\mathbb{H}$ iff f appears in some correspondence rule of a subset H of C, which includes the instructions of target machine $\mathbb{H}$, as well as the fetching and decoding mechanism [H+V89a; b]. We call a term *algorithmic* iff all its operation symbols correspond to algorithmic operations, $+$, $*$, $\times$, ${}^{\times n}$, ${}^{\times *}$, *† and *target* iff it is algorithmic and all its constant symbols corresponds to easy problems. We will denote by $\mathbb{T}_{\mathbb{H}}$ the set of target terms and by $\mathcal{L}_{\mathbb{H}}$ the corresponding *target language*.

As mentioned before, a target machine $\mathbb{H}$ together with a program p (i.e. a target -with respect to $\mathbb{H}$- algorithmic term) realizes a virtual machine $\mathfrak{m}_{p\mathbb{H}}$ which computes the relation $q_{\mu[p]}$. Thus, if $\mathfrak{m}_{p\mathbb{H}}$ is a deterministic machine then problem $\mu[p]$ is deterministic (i.e. the elements of $\mathfrak{P}$ should be symbols denoting deterministic problems and the operation symbols are interpreted in a deterministic fashion). On the other hand, what will be the situation if $\mathfrak{m}_{p\mathbb{H}}$ is a non-deterministic machine? We can say, without loss of generality, that the machine computes a non-empty set Ω of extensions of Skolem functions of the above condition $q_{\mu[p]}$. In general, we can state that the machine $\mathfrak{m}_{p\mathbb{H}}$ computes a set Ω as above. Thus, Ω can be regarded as a solution for problem $\mathbb{P}$, provided that $\mu[p] \lrcorner \mathbb{P}$. This idea of solution is more an extensional one, related to restricting the choice involved in accepting condition $q_{\mathbb{P}}$ as a solution for $\mathbb{P}$. The concept of solution we are interested in is an intensional one, related to the concept of *solving*.

At this point we can consider notions of correctness between specifications, by lifting semantical notions to the syntactical layer. We refer to specifications, rather than to programs, to encompass the entire construction process, throughout specifications and programs. Given terms $\mathcal{F}$ and $\mathcal{G}$ of $\mathcal{L}$, we say that $\mathcal{F}$ is *partially correct* with respect to $\mathcal{G}$ (denoted $\mathcal{F} \leq \mathcal{G}$) iff $\mu[\mathcal{F}] \lrcorner \mu[\mathcal{G}]$, $\mathcal{F}$ *terminates* (denoted $\mathcal{TF}$) iff $\mathcal{V}ib\ \mu[\mathcal{F}]$, and $\mathcal{F}$ is *totally correct* with respect to $\mathcal{G}$ (denoted $\mathcal{F} < \mathcal{G}$) iff $\mathcal{TF}$ and $\mathcal{F} \leq \mathcal{G}$. As we have seen these definitions include the case of non-deterministic programs. Given terms p and $\mathcal{G}$ of $\mathcal{L}$, we say that p is a *solution* for $\mathcal{G}$, with respect to target machine $\mathbb{H}$, denoted $p \underset{\mathbb{H}}{\Leftarrow} \mathcal{G}$, iff $p < \mathcal{G}$ and $p \in \mathbb{T}_{\mathbb{H}}$. Then, a *solution* for a problem $\mathbb{P}$, with respect to $\mathbb{H}$, is a is a target term p such that $\mu[p] \lrcorner \mathbb{P}$ and $\mu[p]$ viable; we denote this by $p \underset{\mathbb{H}}{\Leftarrow} \mathbb{P}$. These definitions of solution capture the idea of *construction* of an object for a target machine.

An important tool for the analysis of the connection between application concept and specification, carried out in the sequel, is the following substitutivity result relating relaxation and solution.

Theorem. $p \underset{\mathbb{H}}{\Leftarrow} \mathcal{F} \wedge \mu[\mathcal{G}] \lrcorner \mu[\mathcal{F}] \to p \underset{\mathbb{H}}{\Leftarrow} \mathcal{G}$ (8)

[1] We actually use variables over problems in our program derivation calculus [HVE89], but we do not need them here

VI. CONNECTING THE THEORETICAL AND OBSERVATIONAL LEVELS OF THE SOFTWARE PROCESS.

In order to analyze the connections among application concepts, specifications, programs, and virtual machines, we need a way to talk formally about application concepts. Although problems are theoretical objects, we have associated, in a purposely vague manner, the application concept with the idea of problem. In our framework, application concepts are objects of $\mathcal{L}_O^*$ whereas problems belong to $\mathcal{L}_T$. In such a context, any attempt to treat applications concepts as problems must be based on an explicit connection.

We have data d and results r in the universe $\mathcal{U}$ on the theoretical level, whereas inputs δ and outputs ρ are in the universe $\mathcal{W}$ on the observational level. The distinction between $\mathcal{U}$ and $\mathcal{W}$ should be kept in mind. The only connection between them is provided by a function $f : \mathcal{W} \rightarrow \mathcal{U}$.

First of all, let us examine the connections between application concepts and problems. It would be desirable to have a function translating application concepts to their theoretical counterpart, i.e. problems, induced by f. Thus, we state:

Postulate. Every application concept $\mathbb{A}$ in $\mathcal{L}_O^*$ is *coextensive* with a problem $C(\mathbb{A})$ in $\mathcal{L}_T$.

We say that an object o of $\mathcal{L}_O^*$ is *coextensive* with an object w of $\mathcal{L}_T$ iff their extensions are *congruent* up to the correspondence rules C[1].

Thus, $\mathbb{A}$ and $C(\mathbb{A})$ are connected by the following natural correspondence rules:

$$(\mathrm{c}_1) \quad d \in D_{C(\mathbb{A})} \leftrightarrow (\exists\delta)(d = f(\delta) \wedge \mathcal{Z}\delta\mathbb{A})$$

$$(\mathrm{c}_2) \quad (\exists\delta)(\mathcal{Z}\delta\mathbb{A} \wedge \mathcal{I}\delta\rho\mathbb{A}) \rightarrow f(\rho) \in R_{C(\mathbb{A})}$$

$$(\mathrm{c}_3) \quad \langle d, r\rangle \in q_{C(\mathbb{A})} \leftrightarrow (\exists\delta)(\exists\rho)(d = f(\delta) \wedge r = f(\rho) \wedge \mathcal{Z}\delta\mathbb{A} \wedge \mathcal{I}\delta\rho\mathbb{A})$$

Notice that we are not claiming knowledge of $C(\mathbb{A})$, but only its existence.

Now, let us make explicit the connection between programs and their virtual machines, which is provided by the subset H of C. If $\mu[f] \in \mathfrak{P}$, target machine $\mathbb{H}$ is able to choose for every data δ, such that $f(\delta) \in D_{\mu[f]}$, a result ρ, so that $\langle f(\delta), f(\rho)\rangle \in q_{\mu[f]}$. Similarly for a program p. Thus, p and $\mathfrak{m}_{pH}$ are connected by the following correspondence rules:

$$(\mathrm{p}_1) \quad \mathcal{A}\mathfrak{m}_{pH}\delta \leftrightarrow f(\delta) \in D_{\mu(p)}$$

$$(\mathrm{p}_2) \quad \mathcal{A}\mathfrak{m}_{pH}\delta \rightarrow [(\exists t)(\mathcal{H}\mathfrak{m}_{pH}\delta t) \leftrightarrow (\exists r)(r \in R_{\mu(p)} \wedge \langle d, r\rangle \in q_{\mu(p)})]$$

$$(\mathrm{p}_3) \quad \mathcal{A}\mathfrak{m}_{pH}\delta \rightarrow [(\exists t)(\mathcal{H}\mathfrak{m}_{pH}\delta t \wedge \rho = \mathfrak{m}_{pH}t(\delta)) \leftrightarrow \langle f(\delta), f(\rho)\rangle \in q_{\mu(p)}]$$

Call $\mathcal{Z}$ the conjunction $\mathrm{c}_1 \wedge \mathrm{c}_2 \wedge \mathrm{c}_3 \wedge \mathrm{p}_1 \wedge \mathrm{p}_2 \wedge \mathrm{p}_3$. We now have $\mathcal{Z} \vdash p \underset{H}{\leftarrow} C(\mathbb{A}) \rightarrow \mathfrak{m}_{pH} \angle \mathbb{A}$ (4).

Recall that reductive statement (1) involves two crucial aspects, namely halting of $\mathfrak{m}_{pH}$ and appropriateness of the outputs. We examine halting first.

VI.1. The need of theoretical reasoning. Now we can proceed to overcome the $\mathcal{L}_O$-undecidability of the halting of $\mathfrak{m}_{pH}$. The crucial difference between $\mathcal{T}p$ and $(\exists t)(\mathcal{H}\mathfrak{m}_{pH}\delta t)$ resides in the fact the former is a predicate introduced into the theoretical language, hence amenable to formal proof, whereas the latter is an

[1] This desideratum can be interpreted as stating that our extended observational language $\mathcal{L}_O^*$ will contain only applications concepts that can be regarded as having a data domain, a result domain and a condition, i.e., our application concepts can be apprehended by means of Polya's three questions. As Ludwig Wittgenstein states in his Tractatus Logico-Philosophicus: "was sich überhaupt sagen läßt, läßt sich klar sagen, und wovon man nicht reden kann, darüber muß man schweigen".

observational formula that is not $\mathcal{L}_O{}^*$-decidable. Thus, one can consider the syntactical object p and try to prove its termination.

Predicates $\mathcal{T}$, of $\mathcal{L}_T$, and $\mathcal{H}$, of $\mathcal{L}_O{}^*$, are connected by the derived rule deduced from $\mathcal{Z}$:

$$(p_4) \quad \mathcal{T}p \rightarrow (\forall\delta)(\mathcal{A}m_{pH}\delta \rightarrow (\exists t)(\mathcal{H}m_{pH}\delta t))$$

We can Skolemize $(\exists t)(\mathcal{H}m_{pH}\delta t)$ by introducing a function symbol ζ so that $\zeta(\delta)$ means "the instant the machine halts after the introduction of the data δ", in the following sense:

$$(p_5) \quad \vdash (\exists t)(\mathcal{H}m_{pH}\delta t) \rightarrow \mathcal{H}m_{pH}\delta\zeta(\delta)$$

Hence, in order to solve our $\mathcal{L}_O{}^*$-undecidability problem, we state:

$$\mathcal{Z} \wedge \mathcal{T}p \vdash (\forall\delta)[\mathcal{E}\delta A \wedge \mathcal{A}m_{pH}\delta \rightarrow \mathcal{H}m_{pH}\delta\zeta(\delta) \wedge (m_{pH} \angle A \rightarrow \mathcal{I}\delta m_{pH}\zeta(\delta)(\delta)A)]$$

Notice that we have eliminated the problem of non-refutability in principle by means of a formal proof of $\mathcal{T}p$, which ensures that $\mathcal{H}m_{pH}\delta t$ will hold for some finite $t = \zeta(\delta)$.

Now we are able to reformulate the naive experiment for being-an-engineering-model, presented above, by instantiating the scheme introduced in section V.

Then, we define H_M: $m_{pH} \angle A$; H_K: $\mathcal{V}\delta\mu[p]$; H_1: $\{\bigwedge_{i=1}^{n} \mathcal{E}\delta_i A, \bigwedge_{i=1}^{n} \mathcal{A}m_{pH}\delta_i\}$. By proving $\mathcal{V}\delta\mu[p]$ and using the previous correspondence rules, we define the observational consequence Oc as $\bigwedge_{i=1}^{n} \mathcal{I}\delta_i m_{pH}\zeta(\delta_i)A$. As we have discussed previously, if it is not the case that $\mathcal{I}\delta_i m_{pH}\zeta(\delta_i)A$ for some $i = 1, \ldots, n$, we are not forced to reject the main hypothesis $m_{pH} \angle A$. We can instead doubt the truth of other statement, such as $\mathcal{E}\delta_i A$, i.e. the fact that this δ_i belongs to the domain of A. On the contrary, if $\bigwedge_{i=1}^{n} \mathcal{I}\delta_i m_{pH}\zeta(\delta_i)A$ holds, we cannot conclude the main hypothesis $m_{pH} \angle A$; we can only consider $\bigwedge_{i=1}^{n} \mathcal{I}\delta_i m_{pH}\zeta(\delta_i)A$ as a good inductive support for $m_{pH} \angle A$.

VI.2. Application concept, specifications, programs and virtual machines. The process of writing a specification $\mathcal{S}pc$ begins with a verbalization $\mathcal{V}_A$, which describes not a single application concept A but a class $K_{\mathcal{V}}$, as we have mentioned. Then, the weakest requirement one can impose on such a specification $\mathcal{S}pc$, in order to ensure that one is solving the correct problem, is that its denotation should be a relaxation of $C(A)$.

Now, from (3) and (4) we derive the *fundamental factorization theorem* for the relation of being-an-engineering-model.

Theorem. $\qquad \mu[\mathcal{S}pc] \sqsupset C(A) \wedge p \underset{H}{\Leftarrow} \mathcal{S}pc \rightarrow m_{pH} \angle A \qquad (5)$

This factorization theorem states formally the general belief of the working software engineer: one can be sure that virtual machine m_{pH} will be an engineering model for application concept A if (i) one constructs a problem specification $\mathcal{S}pc$ whose denotation is a relaxation (up to the correspondence rules) of the application concept A, and (ii) one derives from $\mathcal{S}pc$ a program p, in the target programing language $\mathcal{L}_H$, that is totally correct with respect to $\mathcal{S}pc$.

VII. THE INHERENT NON-MONOTONICITY OF THE SOFTWARE DEVELOPMENT PROCESS.

As we have said, assertion $m_{pH} \angle A$ is synthetic. This synthetic character arises from the fact that A is an extensional object. The only description one has for it is the set of pairs $\langle\delta, \rho\rangle$ such that $\mathcal{E}\delta A$ and $\mathcal{I}\delta\rho A$. In other words, we have no device in $\mathcal{L}_O{}^*$ for describing the extension of A. In fact, similar considerations apply to m_{pH} if we restrict ourselves to the observational language itself. The difference between both objects resides in the fact that $\mathfrak{m}_{pH}$ is an observational object constructed from a term p of the formal language $\mathcal{L}_H$ and the set H of correspondence rules. On the contrary, $\mathbb{A}$ is an observational object expressed directly in the observational language. Note that, although we postulate a theoretical object $C(\mathbb{A})$ corresponding to $\mathbb{A}$, one's knowledge of both objects is only partial. So, one "constructs" a term Spc of $\mathcal{L}$ from an informal verbalization and an incomplete extensional knowledge of $\mathbb{A}$ and, except in trivial cases, one cannot know its complete extension because of its size.

Therefore, one has to accept that the factorization theorem (5) relates two synthetic formulas. This theorem factorizes the synthetic character of $\mathfrak{m}_{pH} \angle \mathbb{A}$ into a synthetic part, $\mu[Spc] \lrcorner C(\mathbb{A})$, and an analytically determinate one, $p \underset{H}{\Leftarrow} Spc$. So, it seems that we can validate $\mu[Spc] \lrcorner C(\mathbb{A})$ by a hypothetico-deductive experiment and prove $p \underset{H}{\Leftarrow} Spc$, instead of validating directly the disposition $\mathfrak{m}_{pH} \angle \mathbb{A}$ as discussed above.

Let us develop an experiment to test the hypothesis $\mu[Spc] \lrcorner C(\mathbb{A})$.

We instantiate $H_M: \mu[Spc] \lrcorner C(\mathbb{A})$; $H_X: \{\mathcal{V}i\delta C(\mathbb{A}), \mathcal{V}i\delta\mu[Spc]\}$; $H_1: E_{\mathbb{A}} \subseteq \{\delta : \delta \in V_{\Omega} \wedge \mathfrak{Z}\delta\mathbb{A}\}$. Now, by using the underlying theory T (the Algebraic Theory of Problems) we derive that testing H_M is equivalent to testing $H_{M1}: D_{C(\mathbb{A})} \subseteq D_{\mu[Spc]}$ and $H_{M2}: q_{\mu[Spc]}\|_{D_{C(\mathbb{A})}} \subseteq q_{C(\mathbb{A})}$. At this point we calculate $\wp_D = \{d : d = f(\delta) \wedge \delta \in E_{\mathbb{A}}\}$, $\wp_R \subseteq \mathfrak{Ran}\, q_{\mu[Spc]}$, and $\wp_q \subseteq q_{\mu[Spc]}\|_{\wp_D}\}$ so that, with $\mathbb{P}^0 = \langle \wp_D, \wp_R, \wp_q \rangle$, we have $\mathcal{V}i\delta\mathbb{P}^0$. Now, we prove (formally) that $\wp_D \subseteq D_{\mu[Spc]}$. The correspondence rules will be $\mathcal{Z}$, and the observational consequence: $\mathcal{O}c: (\forall\delta)(\forall\rho)(\delta \in E_{\mathbb{A}} \wedge f(\rho) \in \wp_R \wedge \langle f(\delta), f(\rho) \rangle \in \wp_q \rightarrow \mathcal{I}\delta\rho\mathbb{A}$.

Now, if we fail to prove $\wp_D \subseteq D_{\mu[Spc]}$ then we must reject H_{M1} and if the experiment falsifies the observational consequence then we must reject H_{M2}. Clearly in either case we must reject H_M. On the contrary, what is the situation when we prove H_{M1} and the experiment does not falsify $\mathcal{O}c$? Obviously, we will not reject H_M, but should we accept it? As we have seen, this is not the case. We can accept only that $\mu[Spc] \lrcorner \mathbb{P}^0$. Then, following the "recipe" of the hypothetico-deductive method we should choose other sets $E_{\mathbb{A}}^1, E_{\mathbb{A}}^2, \ldots, E_{\mathbb{A}}^k$, construct problems $\mathbb{P}^1, \mathbb{P}^2, \ldots, \mathbb{P}^k$, and validate them by using the same experiment scheme. Assume that these experiments do not succeed in falsifying H_M. Then, as working scientists, we will accept H_M, until some new evidence happens to falsify it.

Consider $k+1$ non-rejecting experiments involving problems $\mathbb{P}^0, \mathbb{P}^1, \mathbb{P}^2, \ldots, \mathbb{P}^k$ and let $P = \{\mathbb{P}^i : 0 \leq i \leq k\}$. Assume, for the sake of simplicity, that the problems in P have pairwise disjoint data domains. The only assertion we are entitled to make is that $\mu[Spc]$ is a relaxation of the summation over P. Now, assume we have a program p such that $p \underset{H}{\Leftarrow} Spc$. The previous results guarantee only that $\mathfrak{m}_{pH}$ will be an engineering model of the observational counterpart of this summation. Consider now an experiment with set of data $E_{\mathbb{A}}^{k+1}$ that falsifies H_M. Then, we will have a problem $\mathbb{P}^{k+1}$ for which $D_{\mathbb{P}^{k+1}} \subseteq D_{\mu[Spc]} \vee q_{\mathbb{P}^{k+1}} \subseteq q_{\mu[Spc]}\|_{\wp_D}$ does not hold. A software-engineering interpretation of this disjunction is beyond the scope of this paper; the interested reader should refer to [H+V89a, 89b].

Hence, after being sure of $p \underset{H}{\Leftarrow} Spc$, no matter how much one has validated Spc, $\mathfrak{m}_{pH}$ should be repeatedly validated with respect to $\mathbb{A}$, like any construction in empirical science. In particular, the case of $D_{C(\mathbb{A})} \subseteq D_{\mu[p]}$ failing to hold is one in which $\mathfrak{m}_{pH}$ may fail to halt even though $p \underset{H}{\Leftarrow} Spc$ is guaranteed.

Proving correctness eliminates the possibility of dealing with a program that does not satisfy its specification. The factorization theorem guarantees that if $p \underset{H}{\Leftarrow} Spc$, then the only possible sources of negative experimental outcome is the failure of $\mu[Spc] \lrcorner C(\mathbb{A})$, of some auxiliary hypothesis or of some rule of C. In addition, one must verify termination of p to be "convinced" that virtual machine $\mathfrak{m}_{pH}$ will halt. In fact, one should prove theoretical counterparts, i.e., theoretical predicates related to observational ones by means of rules of C, for every disposition non $\mathcal{L}_O^*$-decidable. Therefore, validation and verification are deeply imbricated and their use in a given order is not only a heuristic strategy but a matter of formal necessity.

Let us go deeper in the analysis of a late rejecting experiment. Before the appearance of problem P^{k+1}, both premises of the factorization theorem were true. But, after acquiring the new knowledge about $\mathbb{A}$ represented by P^{k+1}, the first conjunct, $\mu[Spc] \lrcorner C(\mathbb{A})$, becomes evidently false. This is a flagrant case of non-monotonicity. If one accepts the inherently synthetic character of $\mu[Spc] \lrcorner C(\mathbb{A})$, then one should accept that this non-monotonicity is inherent to the software development process itself. We call this phenomenon *global non-monotonicity*, since it concerns the extreme points of software process.

Let us now analyze, under the circumstances of the preceding paragraphs, the decomposition of the reification leg of the software process (fig I.1) into steps [Tur87]. To accomplish such a decomposition, one bridges the gap between Spc and p by introducing intermediate terms $\mathcal{F}_1, ..., \mathcal{F}_n$ of $\mathcal{L}$, but still not restricted to $\mathcal{L}_H$. In so doing, one creates a sequence of intermediate steps. So, we can write the factorization theorem in the form:

$$\mu[Spc] \lrcorner C(\mathbb{A}) \wedge \mathcal{F}_1 < Spc \wedge \ldots \wedge \mathcal{F}_n < \mathcal{F}_{n-1} \wedge p \underset{H}{\Leftarrow} \mathcal{F}_n \rightarrow \mathfrak{m}_{pH} \angle \mathbb{A}$$

Then, in view of the transitivity of relaxation, we can decompose the preceding statement into:

$$\mu[Spc] \lrcorner C(\mathbb{A}) \wedge \mathcal{F}_1 < Spc \rightarrow \mu[\mathcal{F}_1] \lrcorner C(\mathbb{A}); \ldots; \mu[\mathcal{F}_{n-1}] \lrcorner C(\mathbb{A}) \wedge \mathcal{F}_n < \mathcal{F}_{n-1} \rightarrow \mu[\mathcal{F}_n] \lrcorner C(\mathbb{A});$$
$$\mu[\mathcal{F}_n] \lrcorner C(\mathbb{A}) \wedge p \underset{H}{\Leftarrow} \mathcal{F}_n \rightarrow \mathfrak{m}_{pH} \angle \mathbb{A}.$$

Note that the previous non-monotonicity argument now applies to each one of these statements. So, no matter how "microscopic" are the development steps, non-monotonicity will spread to all of them. *Local non-monotonicity* is the name we give to the fact that non-monotonicity permeates into every development step.

VIII. CONCLUSIONS.

We have analyzed the software development process by using Carnap's Two-level Theory of the Language of Science and the Algebraic Theory of Problems. This analysis has shown that many of Carnap's ideas concerning empirical science shed light on software development. In particular, the need of a theoretical level, distinct from the observational one, has been heavily felt for several reasons.

On the observational level one has application concepts and machines; on the theoretical level one has formal specifications and programs. This separation, in addition to its heuristic value, is a matter of necessity. This necessity arises from the fact that most interesting properties of machines turn out to be dispositions that are non-confirmable or non-refutable in principle. A prime example of non-refutability in principle is the case of machine halting. One has to deal with it on the theoretical level, by proving termination of the corresponding program. Before doing this, one cannot validate programs by means of experiments, since total correctness is neither confirmable nor refutable in principle.

The use of this formal framework has had two main advantages. On the one hand, we have been able to state and establish, in a precise way, some facts that are generally believed on intuitive grounds only. On the other hand, this framework has been helpful in paving the way to some important new facts. Thus, we have proved the synthetic character of correctness with respect to an application concept, as well as the inevitability of validation and verification, and the deep imbrication of both techniques for the analysis of program correctness with respect to the application concept. We have also formally proved that the software development process is inherently non-monotonic at any level of decomposition, in that it presents both global and local non-monotonicity. Therefore, any formalism that purports to describe the software development process must accommodate non-monotonicity.

REFERENCES.

[Car36] Carnap, R., Testability and Meaning. Philosophy of Science, t. 3 (1936), and t, 4 (1937).

[Car56] Carnap, R. The Methodological Character of Theoretical Concepts, in Feigl, H., Scriven, M., :Minnesota Studies in the Philosophy of Science, Minneapolis, 1956.

[EVHV89] Elustondo, P., Veloso, P., A., S., Haeberer, A., M.,Vazquez, L. Program Development in the Algebraic Theory of Problems- XVIII Jornadas Argentinas de Informótica e Investigación Operativa. Buenos Aires. September 1989.

[GTW78] Goguen, J., Thatcher, J., Wagner, E., Wright, J. An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types. Current Trends on Programming Methodology , Vol IV, Yeh, R. (Ed.), Prentice Hall, Englewood Cliffs, N. J., 1978, pp. 80-144.

[H+V89a] Haeberer, A., M., Veloso P., A., S. The Requirement-Specification-Program Triangle: A Theoretical Analysis. Pont. Universidade Católica; Res. Rept. MCC 7/89; Rio de Janeiro,1989.

[H+V89b] Haeberer, A., M., Veloso P., A., S. The Inevitability of program testing - A theoretical analysis.Proc. of the IX Intern. Conf. Chilean Chilean Computer Sci. Soc. July 1989.

[HBV87] Haeberer, A., M., Baum G., Veloso P., A., S. On an algebraic theory of problems and software development. Pont. Universidade Católica; Res. Rept. MCC 2/87; Rio de Janeiro, 1987.

[Hem65] Hempel C. Aspects of Scientific Explanation and Others Essays in the Philosophy of Science. Free Press, New York, 1965.

[HVE89] Haeberer, A., M., Veloso P., A., S., Elustondo P.: Towards a Relational Calculus for Software Construction. Pont. Universidade Católica; Res. Rept. MCC 19/89; Rio de Janeiro,1989. Submitted to Formal Aspects of Computing.

[Leh84] Lehman, M., M. A Further Model of Coherent Programming Process. IEEE Proceedings of Software Process Workshop, UK Feb 1984, IEEE Comp. Soc.,1984.

[Pol57] Polya, G. How to Solve it: a new Aspect of the Mathematical Method. Princeton Univ. Press, Princeton, 1957.

[Ste70] Stegmüller W., Probleme und Resultate der Wissenschaftstheorie und Analytischen Philosophie, Band II: Theorie und Erfahrung. Springer Verlag, Heildèrberg, 1970.

[Tar41] Tarski, A. On The Calculus of Relations. Journal of Symbolic Logic, V. 6, N. 3, September1941.

[T+M87] Turski, W., M., Maibaum, T., S., E.: The Specification of Computer Programs. Addison-Wesley, Wokingham, 1987.

[Vel84] Veloso, P., A., S. Outline of a mathematical theory of general problems. Philosophia Naturalis; 21(2-4), 1984.

[V+H89] Veloso, P., A., S., Haeberer, A., M. Software Development: A Problem-theoretic Analysis and Model. 22nd Hawaii International Conference on System Science, Honolulu, January 1989.

[V+E89] Vazquez, L., A., Elustondo, P., Towards Program Construction on the Algebraic Theory of Problems. M.XVIII Jornadas Argentinas de Informótica e Investigación Operativa. Buenos Aires. September1989.

[Zar88] Zara, A. El Método de Jackson como una instanciación de Divide-and-Conquer inducida por los datos. Informe de trabajo final. Universidad del Centro de la Provincia de Buenos Aires (Argentina), Facultad de Ciencias Exactas, 1988.